BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

BACHELOR'S THESIS

Brno, 2019 Dagmar Wikarská

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS
ÚSTAV TELEKOMUNIKACÍ

CRYPTOGRAPHY ON PROGRAMMABLE SMART CARDS
KRYPTOGRAFIE NA PROGRAMOVATELNÝCH ČIPOVÝCH KARTÁCH

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Dagmar Wikarská**
AUTOR PRÁCE

SUPERVISOR — **doc. Ing. Jan Hajný, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**
Ústav telekomunikací

Studentka: Dagmar Wikarská ***ID:*** 195175
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Kryptografie na programovatelných čipových kartách

POKYNY PRO VYPRACOVÁNÍ:

Téma práce je zaměřeno na implementaci kryptografických algoritmů na platformě čipových karet (OS MultOS). Výstupem práce je funkční implementace protokolu pro řízení přístupu do budov a zadaného revokačního schématu CDH16.

DOPORUČENÁ LITERATURA:

[01] MENEZES, Alfred, Paul C VAN OORSCHOT a Scott A VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[02] MultOS Technical Library [online]. [cit. 2018-09-06]. Dostupné z: https://www.multos.com/developer_centre/technical_library.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRACT

This thesis explores the topic of cryptography schemes using anonymous credentials and revocation. It uses smart cards as anonymous authentication devices with blacklist revocation. Final output of this work is an implementation of a revocation scheme CDH16 on MultOS platform.

Secure authentication devices used in banking, transportation, or building access, enhances anonymity of the user. Implemented protocols could be used as such tools, preventing verifier from knowing irrelevant private information. Added revocation scheme protects from misuse of revoked cards by using blacklisting. Blacklisted cards are blocked and verification fails. This requires update of all verifiers blacklist each epoch.

KEYWORDS

Anonymous credentials, zero knowledge, revocation for ABC, MultOS smart cards, smart cards enhanced security, blacklisting

ABSTRAKT

Tato práce se zabývá problematikou kryptografických schémat využívajících anonymní pověření a revokační schéma. Využívá čipové karty jako anonymní autentizační zařízení s revokací za použití černých listin. Finálním výstupem je implementace revokačního schématu CDH16 na platformě MultOS.

Bezpěčná autentizační zařízení používané v bankovnictví, dopravě či přístupu do budov, zvyšují anonymitu uživatele. Pro zabránění přístupu k privátním datům uloženým na kartě lze využít tyto implementované protokoly. Přidané revokační schéma zabraňuje zneužití revokovaných karet pomocí černé listiny. Karty produkující důkazy nacházající se na takové černé listině, jsou zablokovány a verifikace selže. Tento mechanismus tedy vyžaduje pravidelnou aktualizaci černých listin u všech ověřovatelů pro každou epochu.

KLÍČOVÁ SLOVA

Anonymné pověření, důkazy s nulovou znalostí, revokace pro ABC, MulOS čipové karty, rozšířená zabezpečení čipových karet, černá listina

WIKARSKÁ, Dagmar. *Cryptography on programmable smart cards*. Brno, Rok, 39 p. Bachelor's Thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications. Advised by doc. Ing. Jan Hajný, Ph.D.

ROZŠÍŘENÝ ABSTRAKT

Tato práce se zabývá problematikou kryptografických schémat využívajících anonymní pověření a revokační schéma. Využívá čipové karty jako anonymní autentizační zařízení s revokací za použití černých listin. Finálním výstupem je implementace revokačního schématu CDH16 na platformě MultOS.

Bezpěčná autentizační zařízení používané v bankovnictví, dopravě či přístupu do budov, zvyšují anonymitu uživatele. Pro zabránění přístupu k privátním datům uloženým na kartě lze využít tyto implementované protokoly. Přidané revokační schéma zabraňuje zneužití revokovaných karet pomocí černé listiny. Karty produkující důkazy nacházající se na takové černé listině, jsou zablokovány a verifikace selže. Tento mechanismus tedy vyžaduje pravidelnou aktualizaci černých listin u všech ověřovatelů pro každou epochu.

Základní pilíř této práce tvoří kryptografie na eliptických křivkách. Zachovává totiž bezpečnost standardní kryptografie při použití menšího klíče. Proto je často využívána na čipových kartách disponujících omezenými paměťovými schopnostmi. Operace na eliptických křivkách tvoří základ v obou námi použitých protokolech. Použité schéma anonymního pověření [2] je založena na důkazech s nulovou znalostí. Karta obsahuje několik atributů, které se při verifikaci buď zveřejní, nebo zůstanou ukryty. Zveřejněné atributy obsahují soubor informací, které si karta přeje ověřit. Důkaz platnosti těchto informací se vypočítává z nezveřejněných atributů. Použité revokační schéma [1] přidává k verifikaci dodatečný důkaz. Karta disponuje limitovaným množstvím výzev používaných pro výpočet takového důkazu. To znamená, že v případě využití všech těchto důkazů, musí uživavatel počkat na další epochu. Zopakování identické hodnoty výzvy by znamenalo porušení bezpečnosti. Ověřovatel by si totiž mohl spojit uživatele při dvou rozdílných ověřeních. Nebude zneplatněním takové karty s revokačnou schématem funguje na základě použití černých listin. Pro každou revokovanou kartu revokační autorita zveřejní všechny výzvy, které daná karta umí vypočítat pro danou epochu. Když se karta pokouší o verifikaci, ověřovatel zkontroluje černou listinu. Pokud se na ní výzva použitá pro danou epochu nachází, verifikace je zamítnuta.

Pro výstup této práce jsme vytvořili sloučené schéma. Toto schéma, uvedené na Fig. 5.3, je vytvořeno spojením verifikačního a revokačního schématu. Pro obě schémata jsme následně vytvořili matematické důkazy platnosti. Ty slouží k ověření, že i po provedených změnách důkazy pro obě sloučené schémata platí. Takto upravené a spojené schéma jsme následně implementovali na kartu.

Pro implementaci jsme využili platformu MultOS s verzí 4.21. Vytvořený program je složený ze dvou částí, části pro terminál programovaný v Javě a části pro kartu v C s využitím MultOS knihovnami. Komunikace mezi terminálem a kartou využívala APDU. Java aplikace využívala Netbeans IDE a aplikace pro kartu Eclipse IDE, s nahráváním přes MUtil.

Implementace sestávala z předělání již funkční implementace ověřovacího proprotokolu,

kterou vytvořil Ing. Petr Dzurenda. Tuto aplikaci jsme přepsali, upravili názvy tak, aby části protokolu korespondovaly s upraveným schématem. Takto upravenou ověřovací část jsme následně obohatili o revokační protokol. Jeden ze zajímavých aspektů implementace bylo přidání hodnoty pro počet předešlých verifikaci. Na začátku každé verifikace jsme přidali aktuální epochu. Karta po přijetí těchto hodnot porovnává obdrženou epochu s hodnotou epochy, kterou má uloženou z předešlé verifikace. Pokud se shodují, iteruje počet předešlých verifikaci o jedna. Další část implementace vhodná za zmínění je funkce pro přidání karty na černou listinu. Terminál ověřuje revokační důkaz, ověřovací důkaz a černou listinu. V případě, že se nachází výzva použitá pro výpočet revokačního důkazu na černé listině, tak terminál verifikaci zamítne. Platná verifikace nastane pouze v případě platnosti všech tří podmínek: platné ověření verifikace, revokaci, a výzva nenacházející se na černé listině.

Vytvořená implementace spojeného schématu nebyla úplně úspěšná. Výsledné ověření na kartě se nepovedlo zprovoznit. Všechny ostatní výpočty, které tvořily přibližně devadesát procent implementace, byly však správné. Pro výpočet finálního ověření bylo však nutné, aby všechny mezivýpočty fungovaly najednou. Testovali jsme je kombinovaně, některé však navzájem interferovali a ovlivňovaly si výsledky, způsobem který se nám nepodařilo predikovat. Nakolik chyba způsobující nefunkčnost je velmi malá a může být na příliš možných místech, ani po týdnu nasazení a hledání se nám ji nepodařilo identifikovat.

Všechny mezikroky jsou však ošetřeny a mají vytvořeny individuální funkce pro jejich ověření. Finální verifikace sestává z kombinovaného výpočtu ověřovacího a revokačního důkazu. Za pomoci jedné APDU výměny terminál nastaví parametry nonce a epochu, druhou zahájí výpočet obou důkazů na kartě a následnými dvěma získá od karty vypočtené hodnoty, které následne ověří.

DECLARATION

I declare that I have written the Bachelor's Thesis titled "Cryptography on programmable smart cards" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Bachelor's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

Firstly, I would like to thank my thesis supervisor, doc. Ing. Jan Hajný, PhD., for his professional guidance. Without his insightful tips, this thesis would not be possible. Secondly, I would like to thank Ing. Petr Dzurenda, who has been very helpful to offer tips regarding practical part and coding. Last but not least, I would like to thank my flatmates, family and friends, for their emotional support and patience.

Brno . . . . . .

. . . . . .

author's signature

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Contents

1 Introduction

Twenty-first century is the century of data. Mankind produces more information than at any point in the history before. This saturation naturally creates the need of complex security mechanisms to protect the data. Keystone in the development of modern cryptography may be arguably pinpointed to 1970s. Rise of computer era initiated a breakthrough of both symmetric and public-key cryptography. Nowadays, more secure and complex mechanisms are being created, keeping one step ahead from the capabilities of potential attackers.

To ensure that the user has access to the information, we use authentication, which can be traditionally divided into three categories. Authentication by knowledge, ownership, and inheritance. They can be understood as authentication by something you know, something you have and something you are, respectively. There is also fourth factor, the social network of the user, that is, somebody you know [3]. This thesis explores current trends in the authentication by ownership, using smart card as the physical authentication device.

Authentication by ownership means that the user proves his identity by something what he has. Using smart cards as a physical device for this type of authentication is very suitable, for various reasons. Smard cards are small, use enhanced security layers, and are much less susceptible to attacks than mobile phones or other Internet connectable devices.

Smart cards differ from their simpler counterparts in complexity. They are capable of autonomous computing with the help of embedded microchip. Using cryptographic coprocessor, they can also store not only the private key, but also execute fast algorithms. Throughout the communication, both user and the terminal are required to perform computational tasks defined in the protocols. Security and portability of smart cards provide a fast way to ensure secure transactions, e.g. banking or e-business. This can be used in any system that requires secure authentication [4].

2 Cryptographic preliminaries

Cryptographic preliminaries used in our schemes are needed to be introduced before describing cryptographic protocols themselves. Reason for that is pragmatic. To understand the underlying mechanisms used in our scheme, we first need to look at the basic mechanisms behind the protocols. All of the described preliminaries are used in our scheme, however, we mostly focus on the topics that are crucial to successful revocation scheme implementation. ECC[1] is used throughout all of the protocols, as well as zero knowledge proof. Next, we introduce weak Boneh-Boyen signature[2] which is being used in our revocation scheme (4).

2.1 Elliptic curve cryptography

The biggest advantage of the elliptic curve cryptography is that it reaches required security with smaller key size. This means that 256 bit elliptic curve key is equivalent to 3072 bit RSA key. This is suitable for the usage in lightweight devices.

Elliptic curve cryptography is based on operations on elliptic curves. Elliptic curve is a plane algebraic curve defined by the equation:

$$x^2 + a_1xy + a_3xy = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

An additional requirement is that the curve is smooth, in other words, that the equations 2.2 and 2.3 obtained by differentiating both sides, are not both simultaneously satisfied in the curve. Geometrically this guarantees that the curve has a tangent in every point [5].

$$a_1y = 3x^2 + 2a_2x + a_4 \tag{2.2}$$

$$2y + a_1x + a_3 = 0 \tag{2.3}$$

While having prime number p and a field of integers $\mathbb{F}_p$, elliptic curve group E over $\mathbb{F}_p$ could be defined by the equation:

$$x^2 = x^3 + ax + b \tag{2.4}$$

where a and b satisfy $4a^3 + 27b^2 \not\equiv 0 \pmod{n}$. We get the point on elliptic curve with coordinates (x, y) from equation 2.4 where $x, y \in \mathbb{F}_p$[6]. For cryptography purposes, elliptic curves are most commonly used in a simplified form defined in equation 2.4, rather than general form of equation 2.1. Example of such created

[1] Elliptic Curve Cryptography

[2] *weak Boneh-Boyen signature* (wBB)

group for $y^2 = x^3 - 2x + 2$ over the field $\mathbb{F}_{17}$ contains 21 points which satisfy the equation. These 21 points are:

∞	(0,6)	(0,11)	(1,1)	(1,16)	(5,7)	(5,10)
(6,6)	(6,11)	(7,5)	(7,12)	(9,4)	(9,13)	(10,8)
(10,9)	(11,6)	(11,11)	(14,7)	(14,10)	(15,7)	(15,10)

Curve arithmetic is defined in terms of underlying field operations, the efficiency of which is essential. Efficient curve operations are likewise crucial to performance[6].

There are two basic curve operations. First one is adding two points, described by Fig. 2.1, requires exactly one operation for performing computations. Second operation, multiplication by integer, requires more computational power.

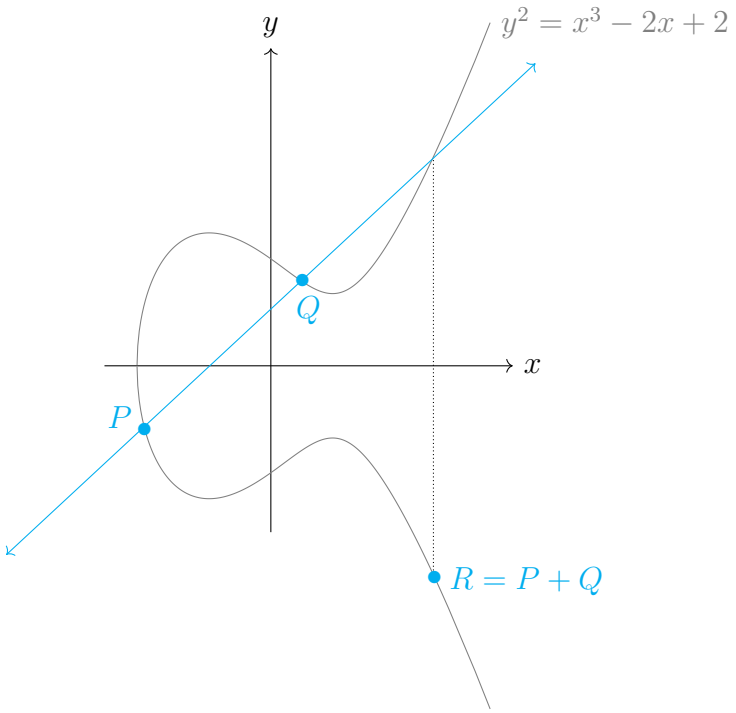

Fig. 2.1: Addition of two points

From the example in Fig. 2.1, we could explain the process of point multiplication. Let's have a point P on the elliptic curve. Drawing a tangent line we get gets us two intersections with an elliptic curve, point P and point $-2P$. Inversing the second, we reach the point $2P$. This is adding the point to itself, or, multiplying P by two. In the ECC, we use multiplication with large sizes of n. Although in the example we display reaching $2P$ by one computation, n being 2, using repetitively

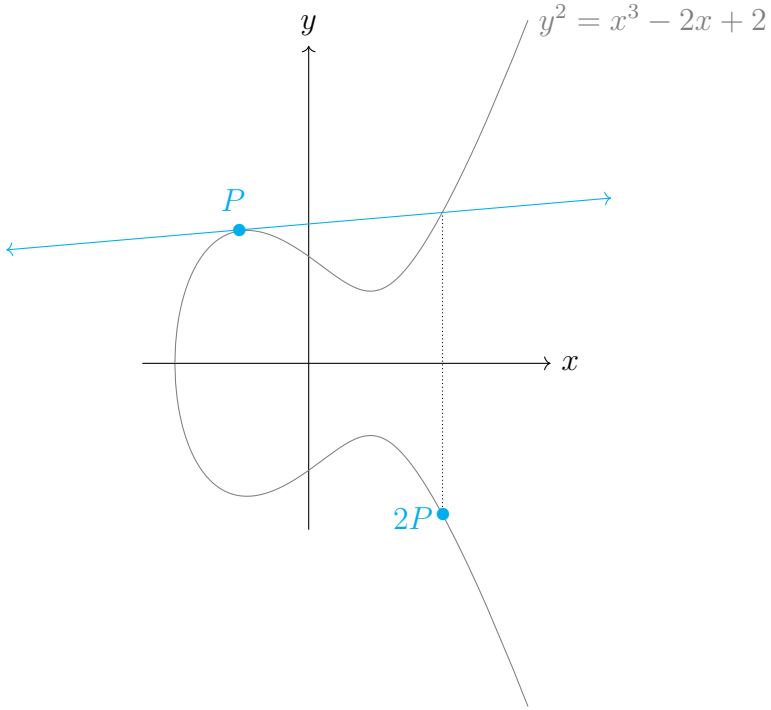

Fig. 2.2: Multiplication of point by integer

tens and thousands of "additions by itself" increases the size of n and reaches substantial security quicker than with conventional discrete logarithm or large number factorization based problem.

2.2 Zero knowledge proof

Fundamental concept of zero knowledge proof is best described by an everyday example. Imagine a card guessing game. One player draws a card from a playing deck and the second one asks: "Is the card red?". "Yes." "Prove it!". Without revealing all the information about the card, including what suit and symbol it is, first player can not prove anything. Unless the players use mathematical mechanism of zero knowledge proof. With this, one can prove legitimacy of certain attribute without disclosing any of the other attributes linked to the first.

The basic idea is to replace "knowledge" by "knowledge about knowledge" [7]. First player's task is not to prove that the card is red, but to prove that he knows the color of the card. He needs to show the proof of his card being red and, ultimately, prove in this guessing game that he did not lie. In the cryptographic algorithms we use in principle the same proof to show that we are the holder of the information. It is the equivalent of showing that we know the status of the card being guessed.

Furthermore, we can extend a zero knowledge proof to **interactive zero knowledge proof** by using an additional challenge-response element. Challenge-response makes user card react to a challenge created by a terminal and use it for computing the response. By this, answer computed by the user is also dependent on the challenge value. Challenge-response is commonly used by itself in various protocol communication schemes. Adding it to the ZKP[3] however enhances the security.

2.3 Weak Boneh-Boyen signature

Weak Boneh-Boyen signatures are essentially unforgeable against a weak chosen message attack under the q-SDH assumption. This assumption uses strong Diffie-Hellman, analogy of strong RSA and was formulated by Boneh and Boyen as a tool to analyse security of wBB in [8] and [9]. The q-SDH assumption is one of the first in a family of new assumptions that have appeared in the context of pairing-based cryptography, and the first of these to be analyzed in the generic group model [10]. wBB can sign only one message, and we use it in our revocation scheme. For general wBB the signature input messages are $m \in \mathbb{Z}_q$, Pk, Sk, and output signature σ). Our method uses wBB without random oracles. Random oracles respond to every input with a random output and are typically used as an assumption proof on the hash function.

[3] *Signature Proof of Knowledge* (SPK)

3 Anonymous credentials

We use smart cards as physical authentication devices. In anonymous credentials schemes, smart card contains certain set of private data called attributes. On request, these attributes are disclosed to the verifier. It is a smart card reader serving as verification authority. However, we do not want to disclose all of our private data when they are not required, so we disclose only chosen subset of our attributes. As this subset changes depending on verification, anonymity of the user is reinforced. For ensuring correct authentication even when half of our attributes stay hidden, we use proof of knowledge system.

Attribute based credentials[1] are used as a digital signature for disclosing only selected information from the stored data. This could be done without revealing anything else by dividing the data into disclosed and undisclosed attributes. Anonymous credential proofs do not require to reveal anything about the stored hidden attributes. Instead, we compute verification proof, which serves as a proof that the carried data satisfy certain properties. These properties are the proof that the card has been issued by trusted issuing authority.

Task of sending disclosed attributes without revealing additional data is already implemented on the card that we use. Used ABC scheme contains two distinct parts, issue protocol and verification protocol. Issue protocol needs to be used only once, as it creates the credentials that are then stored on the card for the entire duration of its life. Verify protocol, on the other hand, is used every time the card wishes to verify some of the attributes.

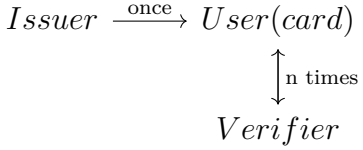

Fig. 3.1: Issue and verify communication scheme

In Fig. 3.1, we can see that the card communicates with two distinct authorities. In our implementation, however, we use one application to serve as both. These mechanisms could be separated if necessary. Same also applies to the revocation scheme, in implementation we use same application for the terminal side. That means, use one terminal serves as three distinct authorities: authority issuing anonymous credentials, authority issuing revocation credentials and having ability to revoke card if necessary, and verification authority.

[1] *Attribute-Based Credentials* (ABC)

3.1 Issue protocol

Issue protocol provides the mechanism for creating attributes. Additionally to these, it computes and stores signature values. Each attribute has its own unique signature, creating a signature-value set. Apart from these signatures linked to certain value, we issue one additional signature. This global signature is computed from special value x_0, which is unique for our issuing authority. It is a signature for card as a whole.

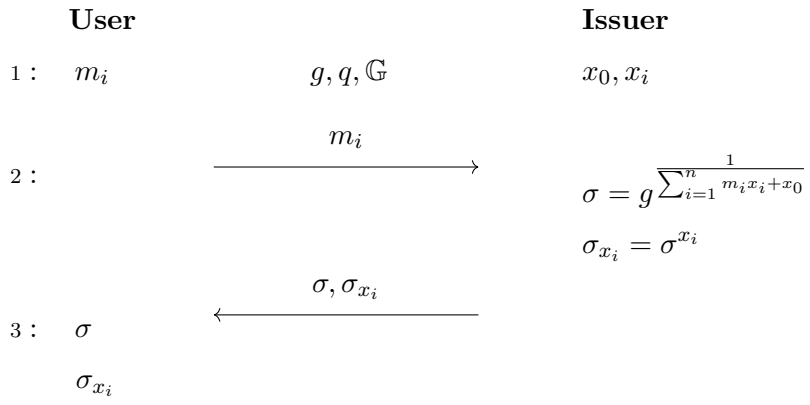

Fig. 3.2: IssueAttribute and ReceiveAttribute protocols [2]

Issue protocol combines two parts, IssueAttribute and RecieveAttribute. Firstly, issuing authority sends value i to the user. This value represents the number of attributes that will be computed. Then, user generates the i number of attributes m_i, one message for each of the attributes. Both i and m_i are stored in the static memory of the card. The attributes are now issued and IssueAttribute part is completed.

Secondly, computed attributes m_i are send back to the Issuer. They need to be send over secure channel, otherwise, with the knowledge of these attributes by the third party, anonymity of the following protocols could be compromised. It is relatively easy to ensure when using smart cards as implementation devices. When issuer receives computed attributes, it creates card signatures σ. Then, for each attribute it computes attribute signature σ_{x_i}.

In the third step, Issuer sends the User computed values σ and σ_{x_i}. Card stores these values and RecieveAttribute protocol is finished.

3.2 Verify protocol

The responsibility of the verification protocol is to reliably check that sent disclosed attributes are correct and issued from trusted authority. Verification protocol needs to ensure the unlinkabillity, untraceabillity and anonymity, as the card user wants to maintain these when communicating with possibly malicious verifier. Unlinkabillity ensures that the attribute signature could not be linked back to the owner, untraceabillity implies impossibility to trace that the verification request came from the same source and anonymity indicates undisclosed identity of the user. For all the cases of usage these smart cards could serve (personal ID, e-ticketing, prepaid pass,...), we need to assume that the verifier could be any terminal. Henceforward it is crucial not to disclose any information regarding the user that he does not wish to disclose.

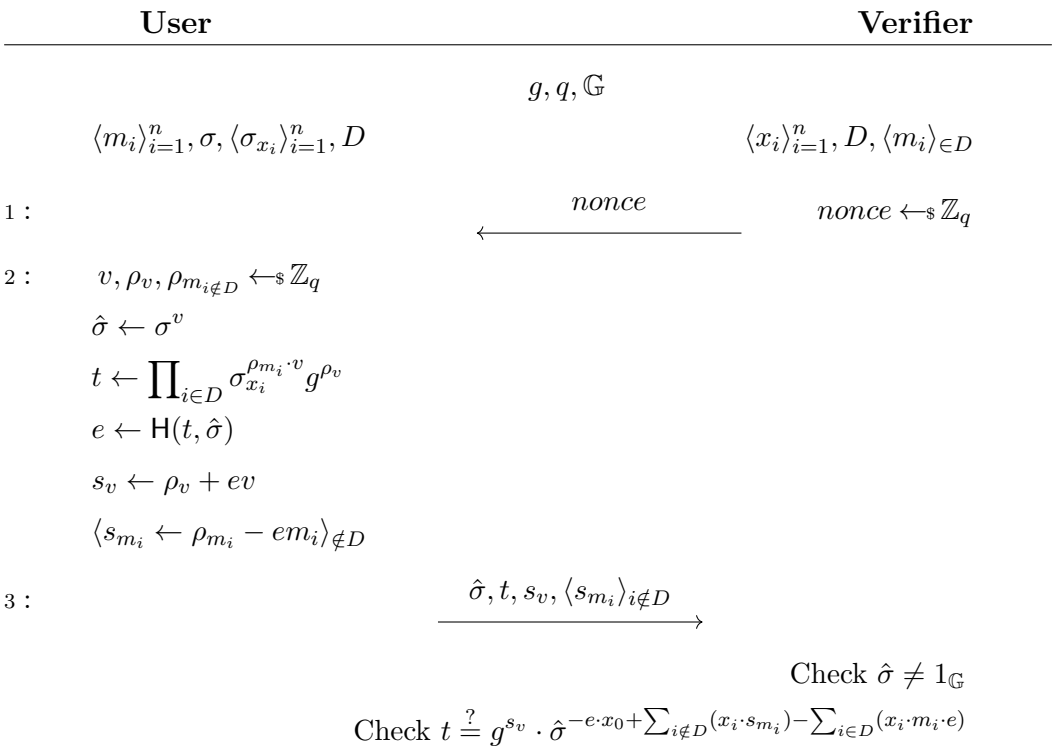

Fig. 3.3: Verification protocol of the KVAC scheme

At the beginning of the protocol, we take the verifier-generated *nonce* (number used only once) and send it to user. In general, *nonce* is a number used as a challenge-response value. User is prompted with a value and needs to use it within the challenge. Then, it sends the answer in the response. Within the Verification

protocol, it is added to the hash creating e. It provides another layer of randomization securing the hash. In the provided implementation, this challenge response mechanism has been omitted. In provided thesis output, we changed that fact and added *nonce* to our scheme.

When user receives the *nonce* value, user computes several auxiliary values and computes the proof. If we used the signature stored in the card for each authentication, untraceabillity attribute would be broken. That's why in second step of the algorithm we create randomized $\hat{\sigma}$ from σ^v. Next, we compute proof t from all the undisclosed attributes that we are not proving in this instance, hash it with other values and let $\langle s_{m_i} \leftarrow \rho_{m_i} - em_i \rangle_{\notin D}$. User sends all the computed values with the proof, namely the randomized signature $\hat{\sigma}$, proof t and set of auxiliary values $s_v, \langle s_{m_i} \rangle_{i \notin D}$ for verification, with the additional s_v computed from the hash of all the remaining, hidden attributes.

Verifier receives the proof set, disclosed attributes, and verifies validity of the response. He checks the proof with the value computed from additional values provided by the user. If computed value matches with t, verification is successful. If not, verifier deems sent disclosed attributes from the card invalid.

4 Revocation scheme

Revocation scheme provides a mechanism to successfully revoke already issued card. Without the revocation scheme, card once issued stays always issued. Imagine an example where you need to ban a driver's license stored on a smart card due to driving under the influence. The smart card contains verifiable driver's license, and we need to find a way of revoking this otherwise valid attribute. While still maintaining security standards. This is done by adding revocation protocol.

RA	**User**	**Verifier**
$m_R \leftarrow_\$ \mathbb{Z}_q$		
$l_1 \dots l_{10} \leftarrow_\$ \mathbb{Z}_q$		
$\alpha_1, \alpha_2 \leftarrow_\$ \mathbb{Z}_q$	$\xrightarrow{m_R, l_1 \dots l_{10}, \alpha_1, \alpha_2}$	
	$r = \alpha_1 l_{1-10} + \alpha_2 l_{1-10}$	
	$c = g^{\frac{1}{r - m_R + \mathsf{H}(epoch)}}$	
	$\pi = SPK(m_R, r)$ $\xrightarrow{\pi, c}$	
		$t_{revoke} \stackrel{?}{=} (gc^{-\mathsf{H}(epoch)})^{-e} c^{s_{m_R}} c^{s_r}$

Fig. 4.1: Revocation protocol for $j = 10$ and $k = 2$ while number of α is k and number of l is j

Before proceeding to describe the revocation protocol, it is worthwhile to mention the Camenisch and Stadler notation [11] used in Fig. 4.1. This shortens the size of the notation for the proof of knowledge protocols, in our instance denoting proof as $SPK(m_R, r)$. Full proof is depicted in figure 4.2. Instead of fully describing computation of proof set with all the auxiliary values, Camenisch and Stadler notation describes the nature of the proof.

$$
\begin{aligned}
\pi = SPK(m_R, r): \quad & \rho_R, \rho_r \leftarrow_\$ \mathbb{Z}_q \\
& t = c^{\rho_{m_R}} c^{\rho_r} \\
& e = \mathsf{H}(t_{revoke}, c) \\
& s_r = \rho_r - e \cdot r \\
& s_{m_R} = \rho_R - e \cdot m_R
\end{aligned}
$$

Fig. 4.2: Camenisch and Stadler notation for our proof of knowledge π

4.1 Revocation protocol description

There are several distinct concepts in this protocol. One of them is incorporation of time element. By including current time, or *epoch*, each proof computed from shared parameters could be used for verification only once for the duration of *epoch*. Revocation authority issues interlinked parameters α and l. In our implementation, we use two values of α and ten values of l. That gives us 10^2 possibilities for computing unique i. Each i could be used only once during the *epoch* to maintain unlinkabillity, giving us 100 unique proofs. If more proofs are required, the number could be increased by expanding size of j or k from Fig. 6.1. Second shared parameter is m_R, which is used for user identity purposes.

Second concept used in this protocol is blacklisting. Revocation authority computes all of the possible combinations of l and *alpha* values for user she is about to revoke. These computed revocation values c values are afterwards published and shared with verifying authorities. Revocation authority has the capability of computing all of the user's proofs because it computes and stores necessary parameters $(m_R, l_1 \dots l_j, \alpha_1 \dots \alpha_k)$ before sending them to the card. In our implemented protocol, we do not use mechanism for blacklisting just certain selected attribute, but the entire card. In the driving license scenario, user would have been revoked due to one blocked attribute, and any verification request from such card would be denied. After some time revocation authority could take off the user from black list, or assign new parameters. This could happen in the example when driver license is revoked for three months and then valid again.

After implementing revocation protocol, verifying authority has two more steps to check before declaring whether the verification is successful or not. This means that for successful verification:

1. user needs to provide proof for verification protocol, where
$$t = g^{s_v} \cdot \hat{\sigma}^{-c \cdot x_0 + \sum_{i \notin D}(x_i \cdot s_{m_i}) - \sum_{i \in D}(x_i \cdot m_i \cdot v)}$$
2. user needs to provide proof for revocation protocol, where
$$t = (gc^{-\mathsf{H}(epoch)})^{-e} c^{s_{m_R}} c^{s_r}$$
3. verifier can not have user's revocation value c on the black list

At the beginning of our protocol described in the Fig. 6.1, revocation authority randomly computes values of m_u, $l_1 \dots l_{10}$ and α_1, α_2. These are subsequently sent through a secure channel to the user. Secondly, for each verification request in *epoch*, user picks unique combination of two values from $\{l_1 \dots l_{10}\}$. With these, $r = \alpha_1 l_{1-10} + \alpha_2 l_{1-10}$ is computed. Thirdly, verifier checks the proof. He generates randomly ρ_{m_R} and ρ_r from the group $\mathbb{Z}_q$. Computes t, hash ch and signatures s_{m_u}, s_i. If $t = (gc^{-\mathsf{H}(epoch)})^{-e} c^{s_{m_R}} c^{s_i}$, proof is successful.

Our protocol lacks mechanism for checking that α and l comes from the trusted *Revocation Authority* (RA). However, this could be maintained by another layer implemented on top of our scheme.

5 Proof of knowledge schemes

For our practical part of the thesis, we implement protocols introduced in previous chapters. However, in order to prevent name clashes and wrong implementation of separate protocols, we first created a new scheme. This scheme consists of the KVAC scheme[2] and revocation protocol[1].

5.1 Primary proof of knowledge scheme

Before starting to merge these protocols together, let's look at fundamental mechanism behind verification protocol. In the most simplified sense, verification is the series of proofs computed by the user and checked by the verifier. The proofs are computed as a way to show that undisclosed attributes are issued by trusted authority. Furthermore, it shows that disclosed attributes are contain a valid and true information about the user.

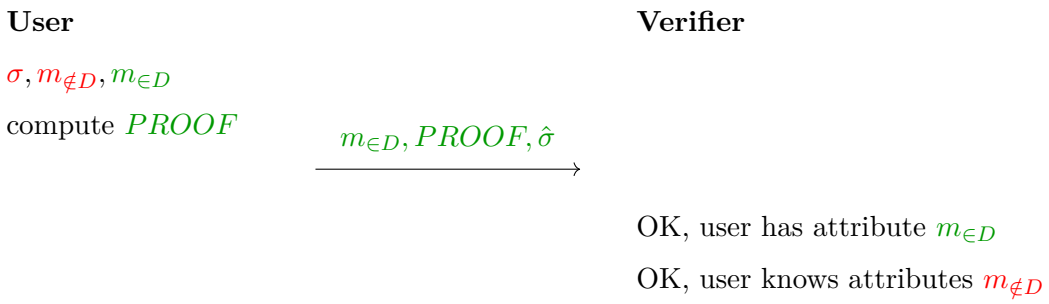

Fig. 5.1: Proof of knowledge scheme without revocation protocol

In the 5.1, we use red to distinguish that values are private. Private values are attributes m_i and a card signature σ. This signature is a unique value that identifies the card. Private attributes are the specific data that card stores, such as age, name, driver's license, or building access. For each verification, we chose which of these values do we wish to disclose for verification, and the subset of private attributes are shared for verification. Hence, values in green are the values shared with a verifier. For verification, card discloses one or more desired attributes. These are sent with the $PROOF$ and a randomized signature $\hat{\sigma}$. $PROOF$ is computed from hidden, undisclosed attributes and serves as the proof of knowledge of these private attributes.

Values that are sent during verification process are $PROOF$, randomized signature $\hat{\sigma}$ computed from σ, and disclosed attribute. Verifier computes the check of these proofs. If $PROOF$ holds, user is verified. If it does not, user is denied.

5.2 Scheme with revocation protocol

Problem with the simple primary proof of knowledge scheme is the fact that when user gets issued, he stays issued forever. There are many cases in which we need to revoke the card, for example in case of the theft, or when some of the attributes on the card stops being true. For this, we add a revocation protocol.

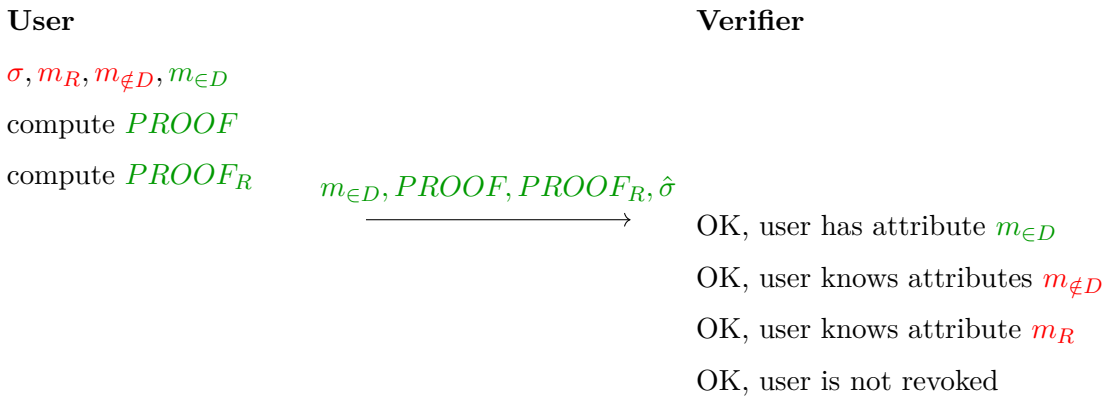

Fig. 5.2: Proof of knowledge scheme with revocation protocol

Revocation protocol takes one of the attributes, m_R, and computes second proof $PROOF_R$. For each verification, verifier compares this proof to the list of revoked cards. If the proof is not there, and at the same time user passed attribute check, verification is successful. $\langle m_i \rangle_{i=1}^{n}$, where m_1 is m_R

5.3 Camenisch and Stadler notation

In the Camenisch and Stadler notation, we can write previously mentioned proofs for revocation and verification as one proof π, where

$$\pi = SPK\{(m_R, m_{\notin D}) : \sigma = g^{\frac{1}{x_0 + x_{\notin D} m_{\notin D} + x_{\in D} m_{\in D} + x_R m_R}} \wedge c = g^{\frac{1}{m_R + v + \mathsf{H}(epoch)}}\}$$

this proof proves the validity of revocation attribute m_R and undisclosed attributes $m_{\notin D}$. Note that this also signifies the fact that the proof is successful only after both revocation and verification succeed.

5.4 Implemented scheme

After knowing the structure of our merged protocol, we proceeded to create the full scheme, displayed in 5.3. Main difference is in changing naming of required values.

These have been changed to better explain what each value represents, such as t used in both protocols has been change to $t_{verifiy}$ and t_{revoke} to differentiate between them. Aside from the name change, these values stay the same. Bigger change has been made in adding the revocation values to hash value e, where the identical hash has been used for both protocols. We added $e = \mathsf{H}(t, c)$ revocation hash values to the randomized verification hash $\mathsf{H}(D, \langle m_i \rangle_{i \in D}, t, \hat{\sigma}, par, ipar, nonce)$. The most crucial change is the fact that we changed computation of s_r from being computed by subtraction $s_r \leftarrow \rho_r - er$ to addition $s_r \leftarrow \rho_r + er$. This has been done to preserve checks, which would not be computed correctly without these changes.

In the implementation scheme, we use m_R as an additional attribute

User	**Verifier**
$m_R, \langle m_i \rangle_{i=1}^{n}, \sigma, l_1 \dots l_{10}, \alpha_1, \alpha_2$	
$r = \alpha_1 l_{1-10} + \alpha_2 l_{1-10}$	
$\sigma_{x_R} = \sigma^{x_R}, \sigma_{x_i} = \sigma^{x_i}$	
$\sigma = g^{\frac{1}{x_0 + x_{\notin D} m_{\notin D} + x_{\in D} m_{\in D} + x_R m_R}}$	
$c = g^{\frac{1}{r - m_R + \mathsf{H}(epoch)}}$	
$v, \rho_v, \rho_r, \rho_{m_R}, \rho_{m_{i \notin D}} \leftarrow_{\$} \mathbb{Z}_q$	
$\hat{\sigma} \leftarrow \sigma^v$	
$t_{verify} \leftarrow g^{\rho_v} \sigma^{\rho_{m_R} \cdot x_R \cdot v} \prod_{i \notin D} \sigma_{x_i}^{\rho_{m_i} \cdot v}$	
$t_{revoke} \leftarrow c^{\rho_{m_R}} c^{\rho_r}$	
$e \leftarrow \mathsf{H}(t_{verify}, t_{revoke}, \hat{\sigma}, c, nonce)$	
$\langle s_{m_i} \leftarrow \rho_{m_i} - e m_i \rangle_{\notin D}$	
$s_v \leftarrow \rho_v + ev$	
$s_{m_R} \leftarrow \rho_{m_R} - e m_R$	
$s_r \leftarrow \rho_r + er$	
$\pi = (\hat{\sigma}, \hat{\sigma}_{x_0}, e, s_{m_{i \notin D}}, s_v, s_{m_R}, s_r)$	
$\xrightarrow{m_{\in D}, \pi, c, \hat{\sigma}, \hat{\sigma}_{x_R}, \hat{\sigma}_{x_i}}$	
	$t_{verify} \stackrel{?}{=} g^{s_v} \cdot \hat{\sigma}_{x_0}^{-e} \cdot \hat{\sigma}_{x_R}^{s_{m_R}} \cdot \prod_{i \notin D} \hat{\sigma}_{x_i}^{s_{m_i}} \prod_{i \in D} \hat{\sigma}^{-e x_i m_i}$
	$t_{revoke} \stackrel{?}{=} (g c^{-\mathsf{H}(epoch)})^{-e} c^{s_{m_R}} c^{s_r}$
	$c \notin$ blacklist

Fig. 5.3: Merged Verify and Revocation protocols, implemented scheme

5.5 Proofs for merged scheme

To ensure that changes we made did not disrupt the functionality of respective protocols, we computed the proofs. For verification proof, we needed to make sure that t_{verify}, which is computed as

$$t_{verify} = g^{\rho_v} \cdot \sigma^{x_R \cdot \rho_{m_R} \cdot v} \cdot \sigma^{x_{i \notin D} \cdot \rho_{m_{i \notin D}} \cdot v} \tag{5.1}$$

results is the same value as the one computed by terminal during the check. To prove this, we used the direct proof and altered only computation used by the terminal. After adjustment, right side is the same as original, card computation of t_{verify}. As we can see, the computation done by the terminal matches the computation done by card, with a distinction of card using hidden secret values and the terminal computing proof by using supplied auxiliary values.

$$\begin{aligned}
t_{verify} &= g^{s_v} \cdot \hat{\sigma}^{-ex_0} \cdot \prod_{\notin D} \hat{\sigma}^{s_{m_i} x_i} \cdot \prod_{\in D} \hat{\sigma}^{-ex_i m_i} \cdot \hat{\sigma}^{s_{m_R} \cdot x_R} \\
&= g^{s_v} \cdot \hat{\sigma}^{-ex_0} \cdot \hat{\sigma}^{s_{m_{\notin D}} \cdot x_{\notin D}} \cdot \hat{\sigma}^{-ex_{\in D} m_{\in D}} \cdot \hat{\sigma}^{(\rho_{m_R} - em_R) \cdot x_R} \\
&= g^{\rho_v + e \cdot v} \cdot \hat{\sigma}^{-ex_0} \cdot \hat{\sigma}^{(\rho_{m_{\notin D}} - e \cdot m_{\notin D}) \cdot x_{\notin D}} \cdot \hat{\sigma}^{-ex_{\in D} m_{\in D}} \cdot \hat{\sigma}^{\rho_{m_R} \cdot x_R - em_R \cdot x_R} \\
&= g^{\rho_v + e \cdot v} \cdot \sigma^{-ex_0 v + \rho_{m_{\notin D}} x_{\notin D} v - em_{\notin D} x_{\notin D} v - ex_{\in D} m_{\in D} v + \rho_{m_R} x_R v - em_R x_R v} \\
&= g^{\rho_v} \cdot g^{e \cdot v} \cdot g^{\frac{-ex_0 v + \rho_{m_{\notin D}} x_{\notin D} v - em_{\notin D} x_{\notin D} v - ex_{\in D} m_{\in D} v + \rho_{m_R} x_R v - em_R x_R v}{x_0 + x_{\notin D} m_{\notin D} + x_{\in D} m_{\in D} + x_R m_R}} \\
&= g^{\rho_v} \cdot g^{\frac{\rho_{m_{\notin D}} x_{\notin D} v - ex_0 v - em_{\notin D} x_{\notin D} v - ex_{\in D} m_{\in D} v + \rho_{m_R} x_R v - em_R x_R v + ex_0 v + em_{\notin D} x_{\notin D} v + ex_{\in D} m_{\in D} v + em_R x_R v}{x_0 + x_{\notin D} m_{\notin D} + x_{\in D} m_{\in D} + x_R m_R}} \\
&= g^{\rho_v} \cdot g^{\frac{+\rho_{m_{\notin D}} x_{\notin D} v + \rho_{m_R} x_R v}{x_0 + x_{\notin D} m_{\notin D} + x_{\in D} m_{\in D} + x_R m_R}} \\
&= g^{\rho_v} \cdot \sigma^{x_R \cdot \rho_{m_R} \cdot v} \cdot \sigma^{x_{\notin D} \cdot \rho_{m_{\notin D}} \cdot v}
\end{aligned}$$

For the revocation protocol, we applied the same method and computed direct proof for the computation of t_{revoke}. We can see that the computation of card and the terminal will create the identical values as well.

$$\begin{aligned}
t_{revoke} &= (gc^{-\mathsf{H}(epoch)})^{-e} c^{s_{m_R}} c^{s_r} \\
c^{\rho_{m_R}} c^{\rho_r} &= g^{-e} c^{e\mathsf{H}(epoch)} c^{s_{m_R}} c^{s_r} \\
c^{\rho_{m_R}} c^{\rho_r} &= g^{-e} c^{e\mathsf{H}(epoch)} c^{\rho_{m_R} - em_R + \rho_r + er} \\
c^{\rho_{m_R}} c^{\rho_r} &= g^{-e} g^{\frac{e\mathsf{H}(epoch) + \rho_{m_R} - em_R + \rho_r + er}{r - m_R + \mathsf{H}(epoch)}} \\
c^{\rho_{m_R}} c^{\rho_r} &= g^{\frac{e\mathsf{H}(epoch) + \rho_{m_R} - em_R + \rho_r + er}{r - m_R + \mathsf{H}(epoch)} - \frac{(r - m_R + \mathsf{H}(epoch)) \cdot e}{r - m_R + \mathsf{H}(epoch)}} \\
c^{\rho_{m_R}} c^{\rho_r} &= g^{\frac{e\mathsf{H}(epoch) + \rho_{m_R} - em_R + \rho_r + er - er + em_R - e\mathsf{H}(epoch)}{r - m_R + \mathsf{H}(epoch)}} \\
c^{\rho_{m_R}} c^{\rho_r} &= g^{\frac{\rho_{m_R} + \rho r}{r - m_R + \mathsf{H}(epoch)}} \\
c^{\rho_{m_R}} c^{\rho_r} &= c^{\rho_{m_R}} c^{\rho_r}
\end{aligned}$$

6 Implementation

There are several OS options for implementing cryptographic schemes on smart cards, all of which come with some problems. They are most often closed systems, and it is impossible to tweak their cryptographic libraries. Because of this, the choice for the right platform and version comes to what do we need to implement. There are three most common OS platform options on the market, JavaCard, BasicCard and MultOS. Their respective support of ECC is presented in Tab. 6.1. For our purposes, we chose MULTOS platform as it provides best options for fundamental ECC operations. Other platforms, such as JavaCard, provide good coverage of fully implemented protocols such as ECDSA, but do not support basic operations in publicly available versions.

Tab. 6.1: ECC support on different smart card platfoms. Excert from [4]

platform	OS version	`ecAdd`	`ecMul`	`ecInv`	ECIES	ECDH	ECDSA
JavaCard	JC 2.2–2.2.2	✗	✗	✗	✗	✓	✓
	JC 3.0.1	✗	✗	✗	✗	✓	✓
	JC 3.0.4	✗	✗	✗	✗	✓	✓
	JC 3.0.5	✓	✓	✗	✗	✓	✓
	JCOP 2.4.1	✓	✓	✗	✗	✓	✓
MultOS	4.2	✓	✓	✓	✓	✓	✓
	4.3.1–4.5.1	✓	✓	✓	✓	✓	✓
Basic Card	ZC5, ZC6	✗	✗	✗	✗	✓	✓
	ZC7, ZC8	✓	✓	✗	✗	✓	✓
.NET Card	.NET 2.0	✗	✗	✗	✗	✗	✗

Note: ✓– algorithm supported, ✗– algoritmus not supported

6.1 Development environments

The practical part of this thesis builds on the implementation of *Keyed-Verification Anonymous Credential Scheme* (KVAC)[2] by Ing. Dzurenda For this, we used card with MultOS version 4.21 and Windows 7 virtual machine with installed and set development tools and environments.

Terminal side has been developed in Netbeans, card computed with MULTOS libraries in C programming language in Eclipse. The tool we used for loading the app on card has been MUtil.

Developing for smart cards is a time consuming process. It requires loading the application instead of simply launching it in IDE. Debugging on card is not available, instead we use error messages in APDU response and creating check instances.

For this reason, apart from using IDEs and development tools, it was crucial to back up the progress using git. With git commits, history and diff tools, we were able to speed the time required to correct the mistake and properly manage the progress.

6.1.1 Netbeans

Terminal side, the Issue, Revocation and Verification autorities have been programmed using Netbeans IDE. Java and curve operation functions are well documented, and we used this advantage to compute the same values on terminal in function such as checkC and checkTRevoke.

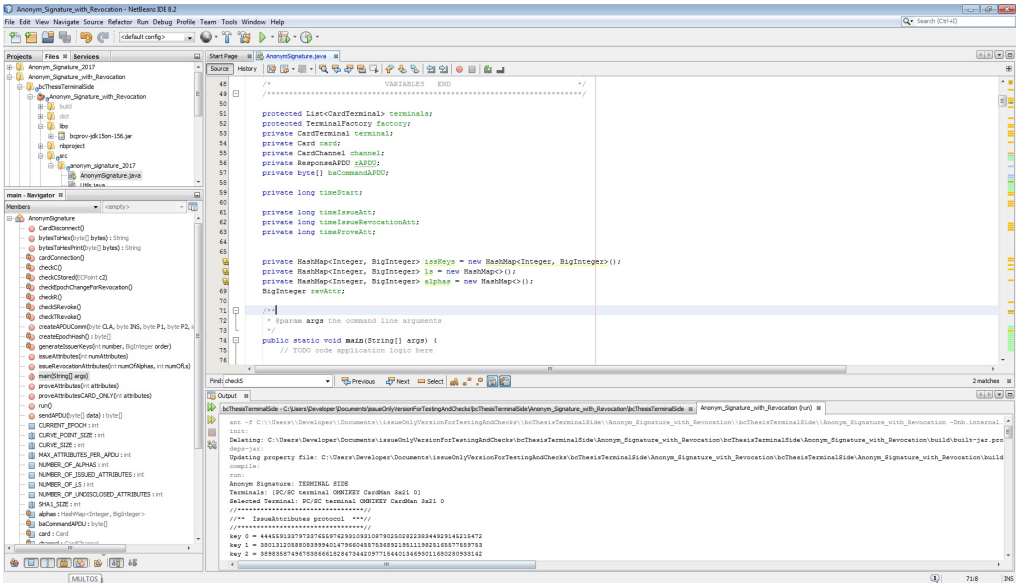

Fig. 6.1: Netbeans IDE

6.1.2 Eclipse

Card side has been developed in Eclipse IDE, which however was not used for launching the application. Instead, we used launch to create .hzx file, each time giving us warning that the IDE couldn't start the application.

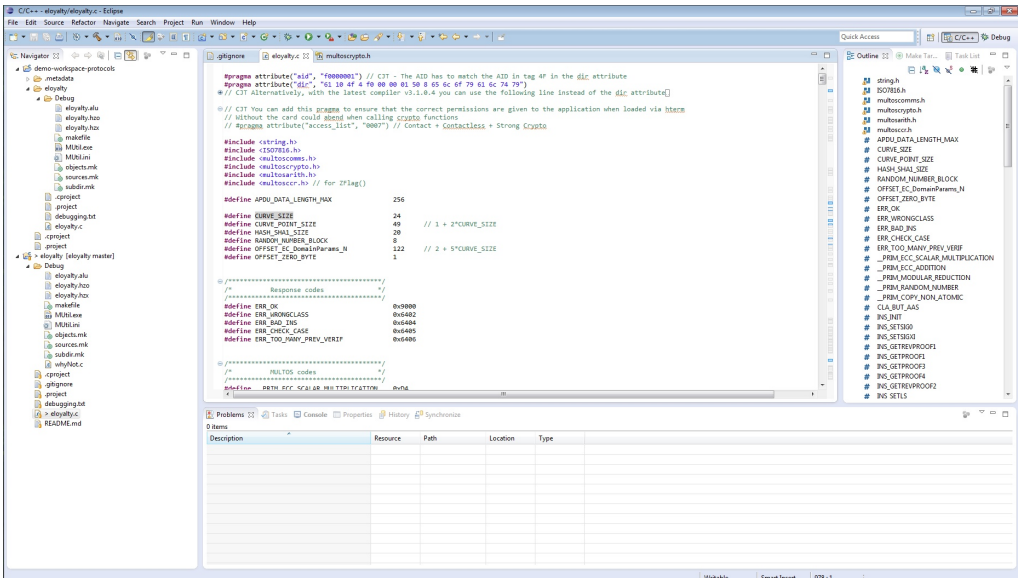

Fig. 6.2: Eclipse IDE

6.1.3 Mutil

After creating .hzx file, we compiled it using terminal command

```
halugen <FILE_NAME>.hzx
```

which created *Application load unit* (ALU) file. After creating the file, we proceeded on loading application in MUtil. We had to define the path of the .alu file, for each load ensuring that the time of compilation is the new one, otherwise it would load previous version. Set the correct AID, and for the bigger application, set the session data size. This is the size in decimal of the RAM storage. To find out what size is required, we checked the .DB size in terminal

```
hls -t <FILE_NAME>.hzx
```

that gave us the required session data size for the application. Without specifying this, launching the application would give us `error 0x8010002f`. This error is also given in case of the wrong APDU response size, explained in section 6.2.

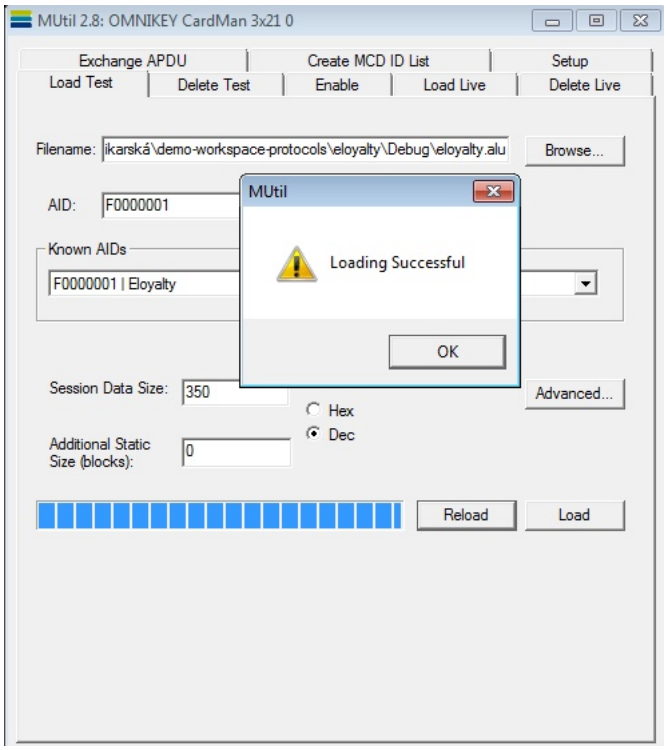

Fig. 6.3: MUtil loading .alu file on card

6.2 Implemented instances of APDU

For implementing working program, we currently explained the choice of platform, and the development requiring two sides: card side in C and terminal in Java. Most crucial part of the equation is however the way these communicate between each other. For this, we need to ensure they are using the same communication protocol. In near field communication, most commonly used is APDU.

Application Protocol Data Unit (APDU) has two types. Communication is always initialized by the terminal, which sends command APDUs. Card answers by sending APDU response. There is no command without the response, and one command APDU always leads to one APDU response.

HEAD					BODY	FOOT
CLA	INS	P1	P2	Lc	DATA	Le
1B	1B	1B	1B	1B	max 255B	1B

Tab. 6.2: Command APDU

Command APDU consists of three parts, head, body and foot. Head contains the class, instance, two parameters, and body length. Class (CLA) specifies which application is going to be run. Instance (INS) is the type within the application. As protocols quite often require more than one APDU exchange, we define these instances. Two parameters, P1 and P2, are optional and could be left empty. They add the parameter within the instance, for example specify number of disclosed attributes in this instance. Lc specifies the size of the body. Because of the limited size of the section Lc that codes length of the carried data, section that contains the data is also limited. The limit is 255 B, which is the largest number Lc could carry. Body contains the customary data which are being sent. Foot is used for containing the length of the response. If set to zero, there will be no response data, if set to number, the response will contain data. Incorrectly defined foot could lead to indexOutOfBoundException, when we are trying to access something that has not been sent.

BODY	FOOT	
DATA	SW1	SW2
max 65536B	1B	1B

Tab. 6.3: Response APDU

Response APDU contains only the data and response codes. Data size could be anything between 0 to 65536, which has been previously defined in Lc.

6.2.1 Error cases

After creating implemented scheme, we have three checks that need to hold in order to successfully verify the card. However, we can not even proceed to these checks when the limit of previous verifications has been reached. We need to ban the card from computing same c as it has before within the epoch, because that would lead to breaking of unlinkabillity. For these purposes, we need to ban the card from computing proofs, when it has reached maximum previous verifications limit. This is done by creating new error case on the card. Card will not compute c and instead, it will send new error case to the terminal. This will let terminal know it has reached the limit of verifications per epoch.

Defining new error case at the beginning of file allows us to create new error case. In our instance we created

```
#define ERR_TOO_MANY_PREV_VERIF    0x6406
```

and for each revocation instance ensured that the card won't compute the proofs. Crucial is the position of the check, as it will not do any operations unless the previous verifications are within the limit.

```
case INS_GETREVPROOF1:
   if (!CheckCase(2)) // case 2 : no data sent, data recieved
      ExitSW(ERR_CHECK_CASE); // error for incorrect case type

   if (numPrevVerifications > MAX_VERIFICATIONS_PER_EPOCH-1) // check if
       limit is not exceeded
      ExitSW(ERR_TOO_MANY_PREV_VERIF); // error exit

   compute_c(); // computes c and stores it in c
   ...
```

Fig. 6.4: Beginning of class instance containing error case

6.3 ECC on MultOS

For our scheme, we could divide implemented operations into two basic categories, those being **integer operations** (integer addition, integer subtraction, integer multiplication, modular inverse) and the **operations on the elliptic curve** (addition of two points, point by integer multiplication). Instead of giving the example for each operation, we refer to the documentation MDRM.pdf.

When starting programming on MultOS platform, we need to think about correct data types for each value that's being implemented. On the card, all of the data based on a BYTE size. We define BYTE in our program as

```
typedef unsigned char BYTE;
```

and it is of size 0 to 255. From BYTE, we create arrays of bytes for of desired value. There are only three types of value used for out implementation, depending on the operation we need to use them for. Simplest one is **??** where in our instance the curve size is 24. The values of this type are used as a integer, for integer operations only.

```
BYTE x[CURVE_SIZE];
```

Fig. 6.5: integer data type

If we need to use a integer value for curve operations, namely point multiplication by integer, we are required to store them as a curve multiplier defined as Figure 6.3.1 due to the fact that the operations used from the MultOS library need to have first byte of the integer set on zero in order to work properly. There is no need to store bigger data types, as the values computed are reduced *mod q* when needed. Especially for integer by point multiplication, reducing the values *mod q* can result in significantly faster computations.

```
typedef struct
{
   BYTE x; // 1
   BYTE ecc_multiplier[CURVE_SIZE]; //24
} ECC_multiplier; // 1 + 24 = 25
```

Fig. 6.6: Curve multiplier data type

The third and the last data type we use is an elliptic curve point Figure 6.3 which also has a first byte empty. From protocol schemes, example of the curve point is t_{revoke}, as t_{revoke} is an addition of two curve points $c^{\rho m_R}$ and $c^{\rho m_r}$

```
typedef struct
{
   BYTE form; // 1 empty byte
   BYTE x[CURVE_SIZE]; // 24 bytes
   BYTE y[CURVE_SIZE]; // 24 bytes
} ECPoint; // 49 bytes ECPoint
```

Fig. 6.7: Curve point data type

In order to perform desired computation, we need to copy the required data on stack, perform operations on stack, and then store the result in desired location. Example of this is a modular inverse, where we take modulus, value, compute on stack, and store the result.

```
// compute modular inverse of z stored in tmp
__push(CURVE_SIZE); //push modulus length CURVE_SIZE
__push(ecDomainParams+OFFSET_EC_DomainParams_N); //push 24 bytes q
__push(CURVE_SIZE); //input length
__push((BYTE*)&z+CURVE_SIZE); // input
__push((BYTE*)&tmp.ecc_multiplier); // address for storing the result, tmp
__code(PRIM, 0xD0, 1); // modular inverse, set to 1 as modulus is prime
```

Fig. 6.8: Performing modular inverse of z

6.3.1 Epoch

Ensuring that there are maximum n verifications per epoch is the key for successful revocation scheme. We had to find the way to create different c for each verification within the same epoch. Creating iteration of αs and ls was one of the most challenging parts of the implementation, making sure that the right combination of of αs and ls is being used. We added an auxiliary value `numPrevVerifications` of a BYTE size. This is being checked at the beginning of each verfication.

```
if (memcmp(epochHash, tmp.ecc_multiplier, CURVE_SIZE) == 0){ // when
    stored and the current epochs are the same
   numPrevVerifications += 1; // iterate plus one
}

else { // epoch hashes do not match
   memcpy((BYTE*)&epochHash, (BYTE*)&tmp.ecc_multiplier, CURVE_SIZE); //
       store new revocation
   numPrevVerifications = 0; // set numPrevVerifications to zero for new
       epoch
}
```

Fig. 6.9: Changing `numPrevVerifications` by comparing stored and currend epoch

At the beginning of the verification, verifier sends the *nonce* value of CURVE_SIZE size and hashed epoch value $\mathsf{H}(epoch)$. For hash, we chose SHA1 as it is of suitable size 20 bytes. This is enough for our computations within CURVE_SIZE (of size 24 bytes), and there are no security and collision concerns. The value is used to represent the certain time range. Crucial assumption is that all of the verifiers use the same time range, so that the attack by repetition can not occur (using

same epoch twice so that the previously computed cs are disclosed). The value `numPrevVerifications` is being used for properly computing r and c values, which are different for each verification.

```
buffer_firstL = buffer_firstL + (numPrevVerifications%10)*CURVE_SIZE;
// (numPrevVerifications % 10) gets second digit of number (0,1,2,3,..)
buffer_secondL = buffer_secondL + (numPrevVerifications/10)*CURVE_SIZE;
// (numPrevVerifications/10) gets first digit of num
    (0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,2,2,2,2,2,...)
```

Fig. 6.10: Set pointer to required l, used in computing $r = \alpha_1 l_{1-10} + \alpha_2 l_{1-10}$

6.3.2 C and blacklisting

To revoke a card, we created a function on terminal side

```
blacklistCardForEpoch(ECCurve curve, ECPoint G)
```

which computes all possible values of c from terminal-created ls and alphas. The function stores these values in a HashMap with values of c as keys, which provides easier comparison with c received from the card.

```
if (everyC_card_epoch.containsKey(c)) // HashMap everyC_card_epoch
{
   // block the card
}
```

7 Conclusion

The main goal of this thesis was to create functional implementation of the revocation protocol CDH16. Unfortunately, the goal has not been completely reached. We have created successful computation of all necessary intermediate steps. After using them within one computation in sequence, they change their value and some parts suddenly fail. The reason for such unsuccessful final check has probably been the interference of different parts of computations. It is hard to pinpoint exact problem due to the fact that during debugging process, error in computation occurs randomly in different places of the computation. It occurs without changing the code or process, just by repetition of launch. Sometimes the computation of c fails, sometimes s_{m_R}, without visible pattern. The other possible reason for this problem is accessing and rewriting wrong value, basically the rogue pointer.

To find the cause of the problem, we have created stand-alone check function for each intermediate value. When computed separately, all of the computations are correct. The separate functions for checking r, c, t_revoke, s_{m_R} and s_r have been initiated and computed correctly. Card contains separate test instances corresponding with created check functions on terminal side.

We have successfully created epoch and computation of unique 100 values for each epoch. We have also added computation of random nonce on terminal. This value is sent to the card and subsequently used for computation of e. We have successfully issued revocation values and attempted to compute revocation proof. The blacklisting function has been implemented successfully.

From the cryptography perspective, we described Issue, Verify and Revoke protocols. We merged these distinct schemes into one, creating one functional merged scheme. We mathematically proved that the verification and revocation proofs still hold after the merge.

Overall, we have created functional program that requires to be optimized. The main hurdle is to find the mistake in computation. As the checks are computed correctly, this may be in a bad usage of temporary values, or bad choice of APDU sequence between terminal and the card. The problem must be by all means little, but there are many possibilities for where it occurred. However, due to the comprehensive checks and functions implemented for development purposes, it could be achieved in the future.

Bibliography

[1] J. Camenisch, M. Drijvers, J. Hajný. Scalable Revocation Scheme for Anonymous Credentials Based on n- times Unlinkable Proofs. *WPES ' 16 Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society. NY, USA: ACM New York* pp. 123-133, 2016. ISBN: 978-1-4503-4569- 9.

[2] J. Camenisch, M. Drijvers, J. Hajný, P. Dzurenda. Fast Keyed-Verication Anonymous Credentials on Standard Smart Cards . *IFIP SEC'19* pp. 123-133, 2019. In print.

[3] J. G. Brainard, A. Juels,R. L. Rivest, M. Szydlo, M. Yung. Fourth-factor authentication: somebody you know. *13th ACM Conference on Computer and Communications Security*, pp. 168–178, 2006. ISBN:1-59593-518-5.

[4] P. Dzurenda, S. Ricci, J. Hajný, L. Malina *Performance Analysis and Comparison of Different Elliptic Curves on Smart Cards*, Table II, 2017. 10.1109/PST.2017.00050.

[5] K. Ruohonen. *Mathematical Cryptology*, pages 82–83, 2014. ISBN 9781501075681. Available from URL: <`https://books.google.sk/books?id=kuLXoQEACAAJ`>

[6] D. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004. ISBN 0-387-95273-X.

[7] U. Feige, A. Fiat, A. J. Shamir. Zero-Knowledge Proofs of Identity *Cryptology*, 1: 77, 1988. Available from URL: <`https://doi.org/10.1007/BF02351717`>.

[8] D. Boneh, X. Boyen. Short signatures without random oracles. *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of Lecture Notes in Computer Science, pages 56–73. Springer, 2004. Available from URL: <`http://crypto.stanford.edu/~xb//eurocrypt04a/`>

[9] D. Boneh, X. Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of Cryptology*, volume 21, pages 149–177, 2008. ISSN 0933-2790

[10] D. Jao, K. Yoshida. Boneh-Boyen Signatures and the Strong Diffie-Hellman Problem. *Pairing-Based Cryptography – Pairing 2009*. ISBN 978-3-642-03298-1.

[11] J. Camenisch, M. Stadler. Efficient group signature schemes for large groups. *Advances in Cryptology – CRYPTO '97*, pages 410–424, 1997. ISBN 978-3-540-63384-6.

[12] G. Seroussi. Elliptic curve cryptography. *Information Theory and Networking Workshop* Metsovo, Greece, 1999. DOI: 10.1109/ITNW.1999.814351. Available from URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=814351&isnumber=17592>.

List of symbols, physical constants and abbreviations

ABC	Attribute-Based Credentials
ALU	Application load unit
APDU	Application Protocol Data Unit
DES	Data Encryption Standard
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
KVAC	Keyed-Verification Anonymous Credential Scheme
RA	Revocation Authority
SPK	Signature Proof of Knowledge
wBB	weak Boneh-Boyen signature

List of appendices

Implemented scheme project for terminal side including AnonymSignature.java
Implemented scheme project for card side including eloyalty.c